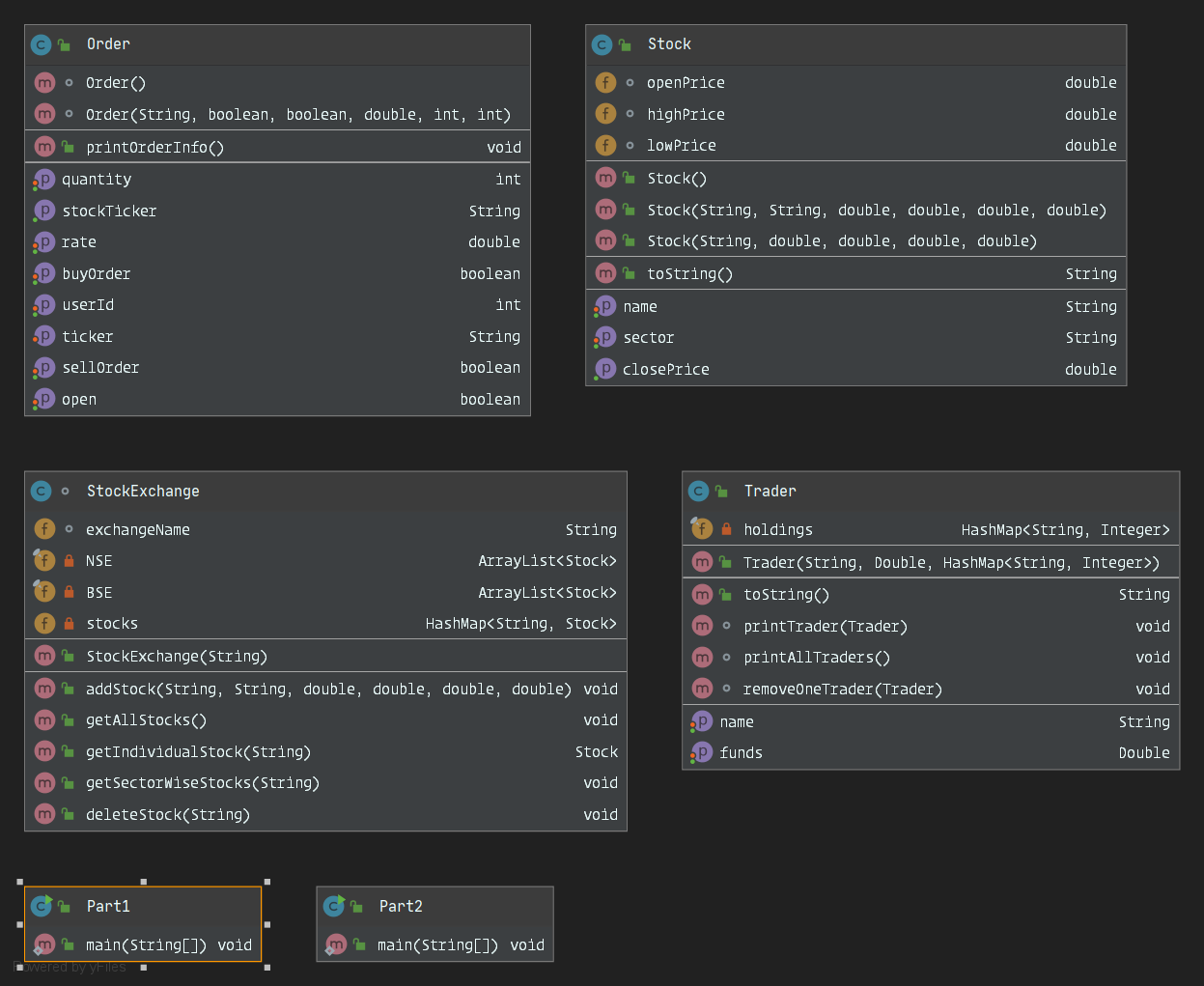
**GRADED LAB ASSIGNMENT 2**

SIDDHANT MITTAL

1910110388

**UML DIAGRAM:**



**CODE:**

**STOCK CLASS**

package com.assignment;

// creating the class for a stock with following parameters

public class Stock{

//variables required for creating a stock

String scrip;

String sector;

double openPrice;

double highPrice;

double lowPrice;

double closePrice;

// instantiating a new empty stock object

public Stock(){

this.sector="";

this.openPrice=0;

this.highPrice=0;

this.lowPrice=0;

this.closePrice=0;

}

// constructor for adding the following data

public Stock(String s,String sec, double o, double h,double l,double c) {

// adding scrip present at index 0 of array

this.scrip = s;

// adding sector

this.sector = sec;

// adding openPrice

this.openPrice = o;

// adding highPrice

this.highPrice = h;

// adding lowPrice

this.lowPrice = l;

// adding closePrice

this.closePrice = c;

}

//constructor for stock class to create a new stock for given ticker in

public Stock(String sec,double o,double h,double l,double c){

this.sector = sec;

this.openPrice = o;

this.highPrice = h;

this.lowPrice = l;

this.closePrice = c;

}

//getter method for stock name

public String getName() {

return scrip;

}

//setter method for stock name

public void setName(String scrip) {

this.scrip = scrip;

}

//getter method for sector

public String getSector() {

return sector;

}

//setter method for sector

public void setSector(String sector) {

this.sector = sector;

}

//getter method for closePrice

public double getClosePrice(){

return closePrice;

}

//toString() method to print details

@Override

public String toString() {

return "O: " + openPrice +

", H: " + highPrice +

", L: " + lowPrice +

", C: " + closePrice;

}

}

**STOCKEXCHANGE CLASS:**

package com.assignment;

import java.util.\*;

// Two StockExchanges namely NSE and BSE

// Stocks will be stored in each accordingly

// A stock can be added to NSE,BSE or both.

// Create a new stock and add it to StockExchange

class StockExchange{

//ticker for the stock

String exchangeName;

// arraylist for NSE stockExchange

private final ArrayList<Stock> NSE = new ArrayList<Stock>();

// arraylist for BSE stockExchange

private final ArrayList<Stock> BSE = new ArrayList<Stock>();

// to add stocks to stockExchange

private HashMap<String,Stock> stocks = new HashMap<String,Stock>();

//creating/initiating a new stockExchange namely NSE/BSE

public StockExchange(String name) {

this.stocks = new HashMap<String, Stock>();

this.exchangeName = name;

}

// add a new stock to the stockExchange

public void addStock(String scrip, String sect, double open, double high, double low, double close) {

Stock st = new Stock(sect, open, high, low, close);

stocks.put(scrip, st);

System.out.println("Added scrip: " + scrip);

}

// returns all stocks registered in the stockExchange

public void getAllStocks() {

System.out.println("Scrips:");

for (String tick : stocks.keySet()) {

String temp = "Scrip : " + tick + ", Sector : " + stocks.get(tick).getSector() + ", ";

temp += stocks.get(tick).toString();

System.out.println(temp);

}

}

// to return stock of a particular ticker

public Stock getIndividualStock(String ticker){

return stocks.get(ticker);

}

// to return all stocks sector-wise

public void getSectorWiseStocks(String sect) {

System.out.println("Scripts listed in " + sect + ":");

for (String tick : stocks.keySet()) {

if (stocks.get(tick).getSector().equals(sect)) {

String temp = "Scrip : " + tick + ", ";

temp += stocks.get(tick).toString();

System.out.println(temp);

}

}

}

// delete a stock for a given ticker

public void deleteStock(String ticker) {

//Delete Stock from here

System.out.println("Deleted scrip: " + ticker);

stocks.remove(ticker);

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TRADER CLASS:**

package com.assignment;

import java.util.HashMap;

//Trader class for registering a new trader to ouy system

public class Trader {

//declaring the required variables

private String name;

private Double funds;

private final HashMap<String, Integer> holdings;

// constructor for trader class

public Trader(String name, Double funds,HashMap<String,Integer> holding) {

this.name = name;

this.funds = funds;

this.holdings=holding;

System.out.println("Added new user: " + this.name);

}

// getter method for name

public String getName() {

return name;

}

// setting the name for the trader

public void setName(String name) {

this.name = name;

}

// getter method for funds

public Double getFunds() {

return funds;

}

// setter method for funds

public void setFunds(Double funds) {

this.funds = funds;

}

// toString() method for printing the details

@Override

public String toString() {

return "Trader { " +

"name='" + name + '\'' +

", funds=" + funds +

", holdings=" + (holdings == null ? "None" : holdings) +

" }";

}

// method to print trader info

void printTrader(Trader t){

System.out.println("User: " + t.name + ", Funds: " + t.funds + " Holdings: " + t.holdings);

}

// method to print all the registered traders

void printAllTraders(){

System.out.println("User: " + this.name + ", Funds: " + this.funds + " Holdings: " + this.holdings);

}

//method to remove a trader

void removeOneTrader(Trader t){

System.out.println("Removed trader!");

}

}

**ORDER CLASS:**

package com.assignment;

//Order class for creating and instantiating a new order!

public class Order {

private String Ticker;

private boolean buyOrder;

private boolean sellOrder;

private double rate;

private boolean isOpen;

private int quantity;

private int userId;

// setting up a new order

Order() {

this.Ticker = "";

this.buyOrder = false;

this.sellOrder = false;

this.rate = 0;

this.isOpen = false;

this.quantity = 0;

this.userId = -1;

}

// creating a new order for the following stock

Order(String tick, boolean buyOrder, boolean sellOrder, double rate, int qty, int id) {

this.Ticker = tick;

this.buyOrder = buyOrder;

this.sellOrder = sellOrder;

this.quantity = qty;

this.userId = id;

this.isOpen = true;

this.rate = rate;

}

//method to set the rate

public void setRate(double rate) {

this.rate = rate;

}

//method to return rate

public double getRate() {

return rate;

}

//check if order is open

public boolean isOpen() {

return isOpen;

}

//set the order open property

public void setOpen(boolean open) {

isOpen = open;

}

//setting up the stock ticker

public void setTicker(String stockTicker) {

Ticker = stockTicker;

}

//setting the type of order as BUY

public void setBuyOrder(boolean buyOrder) {

this.buyOrder = buyOrder;

}

//setting the type of order as SELL

public void setSellOrder(boolean sellOrder) {

this.sellOrder = sellOrder;

}

//setting order quantity

public void setQuantity(int quantity) {

this.quantity = quantity;

}

//setting up userid for who placed the order

public void setUserId(int orderMadeBy) {

this.userId = orderMadeBy;

}

//getting the stock ticker

public String getStockTicker() {

return Ticker;

}

//type of order

public boolean isBuyOrder() {

return buyOrder;

}

//type of order

public boolean isSellOrder() {

return sellOrder;

}

//return quantity

public int getQuantity() {

return quantity;

}

//who created the order

public int getUserId() {

return userId;

}

//method to print order info

public void printOrderInfo() {

String t = "";

if (this.buyOrder) {

t = "BUY";

} else if (this.sellOrder) {

t = "SELL";

}

System.out.println(t + " " + this.Ticker + " Qty: " + this.quantity + " Rate: " + this.rate + "\n");

}

}

**MAIN CLASS FOR PART1 OF ASSIGNMENT**

package com.assignment;

import java.io.\*;

import java.util.\*;

/\*

In the indian stock market there are namely 2 stockExchanges NSE and BSE in which the stock are registered individually.

We will add/delete the stocks with following properties ticker,sectors,O,H,L,C

We will add/delete the trader having a uniqueId,name,funds and his holdings.

We will place orders buy/sell and validate them

We will check for the executed transactions when askPrice<=bidPrice and

considering other parameters too.

\*/

public class Part1 {

public static void main(String[] args) {

try{

System.out.println("Part 1: " +"\n" + "Instantiating my MiniBrokerageSystem!");

//created the objects for stockExchange NSE and BSE

StockExchange NSE = new StockExchange("NSE");

StockExchange BSE = new StockExchange("BSE");

//extracting data from the file

ArrayList<String> ac = new ArrayList<String>();

ArrayList<String> au = new ArrayList<String>();

ArrayList<String> po = new ArrayList<String>();

//reading from the input.txt file given

File F = new File("input.txt");

Scanner reader = new Scanner(F);

while(reader.hasNextLine()){

String Data = reader.nextLine();

if(Data.startsWith("Add scrip:")){

ac.add(Data);

}

if(Data.startsWith("Add user:")){

au.add(Data);

}

if(Data.startsWith("Place order,")){

po.add(Data);

}

}

reader.close();

//extracting the stock data from the given input file

int k = 0;

String[] scrip = new String[20];

String[] sec = new String[20];

Double[] o = new Double[20];

Double[] h = new Double[20];

Double[] l = new Double[20];

Double[] c = new Double[20];

for(String i: ac){

scrip[k] = (i.substring(i.indexOf(":")+2,i.indexOf(",")));

sec[k] = (i.substring(i.indexOf("sector:")+8,i.indexOf(" O")-1));

o[k] = Double.parseDouble(i.substring(i.indexOf("O:")+2,i.indexOf(" H")-1));

h[k] = Double.parseDouble(i.substring(i.indexOf("H:")+2,i.indexOf(" L")-1));

l[k] = Double.parseDouble(i.substring(i.indexOf("L:")+2,i.indexOf(" C:")-1));

c[k] = Double.parseDouble(i.substring(i.indexOf("C:")+2));

k++;

}

System.out.println();

//register the following stocks in system

for(int i=0;i<6;i++){

NSE.addStock(scrip[i],sec[i],o[i],h[i],l[i],c[i]);

}

String[] username = new String[10];

Double[] fundss = new Double[10];

String[] str = new String[10];

//extracting the required trader data from the file

for(String i: au){

username[k] = (i.substring(i.indexOf(":")+2,i.indexOf(",")));

fundss[k] = Double.parseDouble(i.substring(i.indexOf("funds:")+6,i.indexOf(" h")));

str[k] = i.replace(",","");

str[k] = str[k].substring(str[k].indexOf("holding: ")+9);

str[k] = str[k].replace("{","");

str[k] = str[k].replace("}","");

str[k] = str[k].replace(" ","");

k++;

}

//extracting the place order related data from the file

String[] user = new String[100];

String[] type = new String[100];

String[] scr = new String[100];

Double[] rate = new Double[100];

Integer[] qty = new Integer[100];

for(String i: po){

user[k] = i.substring(i.indexOf("user: ")+6,i.indexOf(" type")-1);

type[k] = i.substring(i.indexOf("type: ")+5,i.indexOf(" scrip")-1);

scr[k] = i.substring(i.indexOf("scrip: ")+7,i.indexOf(" qty")-1);

qty[k] = Integer.parseInt(i.substring(i.indexOf("qty:")+4,i.indexOf(" rate")-1));

rate[k] = Double.parseDouble(i.substring((i.indexOf("rate: ")+5)));

k++;

}

System.out.println();

//registering the user to the system and instantiating their accounts

ArrayList<Trader> newTrader = new ArrayList<Trader>();

HashMap<String, Integer> holdings = new HashMap<String, Integer>();

newTrader.add(new Trader(username[6], fundss[6], holdings));

HashMap<String, Integer> holdings1 = new HashMap<String, Integer>();

holdings1.put("INFY", 10);

holdings1.put("TCS", 5);

holdings1.put("SBI", 20);

newTrader.add(new Trader(username[7], fundss[7], holdings1));

HashMap<String, Integer> holdings2 = new HashMap<String, Integer>();

holdings2.put("SBI", 100);

holdings2.put("M&M", 20);

newTrader.add(new Trader(username[8], fundss[8], holdings2));

HashMap<String, Integer> holdings3 = new HashMap<String, Integer>();

holdings3.put("INFY", 20);

holdings3.put("M&M", 25);

holdings3.put("SBI",25);

newTrader.add(new Trader(username[9], fundss[9], holdings3));

//setting up the trader id and adding the trader to the stockExchange for placing orders

int traderID = 1;

HashMap<Integer, Trader> traders = new HashMap<Integer, Trader>();

for (Trader i : newTrader) {

traders.put(traderID, i);

traderID++;

}

System.out.println();

//Stock Market Opens!

System.out.println("Market Opens:");

//Creating our OrderBook using Arraylist which will contain all the accepted orders

ArrayList<Order> OrderBook = new ArrayList<Order>();

for(int j=0;j<user.length;j++){

//Placing our order

if(user[j]!=null && type[j]!=null && rate[j]!=null && scr[j]!=null && qty[j]!=null) {

Order newOrder;

String userName = user[j];

boolean isBuy;

boolean isSell;

if (type[j].equals(" buy")) {

isBuy = true;

isSell=false;

}

else {

isSell = true;

isBuy= false;

}

String tick = scr[j];

double givenRate = rate[j];

int stockQty = qty[j];

boolean lowerCircuit = false;

boolean upperCircuit = false;

boolean notEnoughFunds = false;

int userId = -1;

//checking if the following user exists and extracting his id

for (int i : traders.keySet()) {

if (traders.get(i).getName().equals(userName)) {

userId = i;

break;

}

}

if (userId == -1) {

System.out.println("This user is not registered ! Please Try Again");

}

//checking if the user has enough funds to place order

double userFunds = traders.get(userId).getFunds();

if (userFunds < stockQty \* givenRate) {

notEnoughFunds = true;

}

if (notEnoughFunds) {

if (isBuy)

System.out.println("Order rejected for " + "user: " + traders.get(userId).getName() + ", type: buy" + ", scrip: " + tick + ", qty: " + stockQty + ", rate: " + givenRate + ", due to insufficient funds!");

}

//extracting the stock and checking if the stock exists or not

Stock s = NSE.getIndividualStock(tick);

if (s == null) {

System.out.println("The given Stock does not exist!!");

}

assert s != null;

//getting the last close price of stock

double lastClosePrice = s.getClosePrice();

//calculating the Upper Circuit Price : LastClosePrice + 10\*LastClosePrice/100

double upperCircuitPrice = 10 \* lastClosePrice / 100 + lastClosePrice;

//Calculating the Lower Circuit Price : LastClosePrice - 10\*LastClosePrice/100

double lowerCircuitPrice = lastClosePrice - 10 \* lastClosePrice / 100;

if (givenRate < lowerCircuitPrice) {

lowerCircuit = true;

}

if (givenRate > upperCircuitPrice) {

upperCircuit = true;

}

//checking the lower circuit rate

if (lowerCircuit) {

if (isBuy)

System.out.println("Order rejected for " + "user: " + traders.get(userId).getName() + ", type: buy" + ", scrip: " + tick + ", qty: " + stockQty + ", rate: " + givenRate + ", due to lower circuit violation!");

else

System.out.println("Order rejected for " + "user: " + traders.get(userId).getName() + ", type: sell" + ", scrip: " + tick + ", qty: " + stockQty + ", rate: " + givenRate + ", due to lower circuit violation!");

}

//checking the upper circuit rate

if (upperCircuit) {

if (isBuy)

System.out.println("Order rejected for " + "user: " + traders.get(userId).getName() + ", type: buy" + ", scrip: " + tick + ", qty: " + stockQty + ", rate: " + givenRate + ", due to upper circuit violation!");

else

System.out.println("Order rejected for " + "user: " + traders.get(userId).getName() + ", type: sell" + ", scrip: " + tick + ", qty: " + stockQty + ", rate: " + givenRate + ", due to upper circuit violation!");

}

//If the user passes all the above conditions then the order is valid and added to the orderBook according to its type(buy/sell).

if (isBuy) {

if (!(lowerCircuit || upperCircuit || notEnoughFunds)) {

System.out.println("Order placed for " + "user: " + traders.get(userId).getName() + " type: buy" + " scrip: " + tick + " qty: " + stockQty + " rate: " + givenRate);

newOrder = new Order(tick, isBuy, isSell, givenRate, stockQty, userId);

OrderBook.add(newOrder);

}

} else if (isSell) {

if (!(lowerCircuit || upperCircuit)) {

System.out.println("Order placed for " + "user: " + traders.get(userId).getName() + " type: sell" + " scrip: " + tick + " qty: " + stockQty + " rate: " + givenRate);

if(userId==1)

holdings.put(tick,stockQty);

newOrder = new Order(tick, isBuy, isSell, givenRate, stockQty, userId);

OrderBook.add(newOrder);

}

}

}

}

System.out.println();

//Printing our OrderBook if any orders exist!!

if(OrderBook.size() !=0) {

System.out.println("OrderBook: ");

for (Order i : OrderBook) {

if (i.isBuyOrder()) {

System.out.println("Buy order " + i.getStockTicker() + ":" + i.getQuantity() + " at " + i.getRate());

}

}

for (Order i : OrderBook) {

if (i.isSellOrder()) {

System.out.println("Sell order " + i.getStockTicker() + ":" + i.getQuantity() + " at " + i.getRate());

}

}

}

System.out.println();

//extracting the buy and sell orders separately for verifying the execution of order!

ArrayList<Order> sell = new ArrayList<Order>();

ArrayList<Order> buy = new ArrayList<Order>();

//separating the buy and sell orders

for(Order i: OrderBook){

if(i.isSellOrder()) {

if (i.isOpen()) {

sell.add(i);

}

}

if(i.isBuyOrder()){

if(i.isOpen()){

buy.add(i);

}

}

}

//performing the executed transaction method

int f =0;

System.out.println("Executed Transaction: ");

for(Order i: sell){

for(Order j: buy){

if(i.getStockTicker().equals(j.getStockTicker())){

String ss = i.getStockTicker();

if(i.getStockTicker().equals(ss)){

double askPrice = i.getRate();

double bidPrice = j.getRate();

double quantity = Math.min(i.getQuantity(),j.getQuantity());

if(askPrice<=bidPrice){

System.out.println(quantity + " qty of scrip:" + i.getStockTicker() + " sold for INR " + i.getRate() + "; " + "Buyer: " + traders.get(j.getUserId()).getName() + ", Seller: " + traders.get(i.getUserId()).getName() );

int qty1 = j.getQuantity();

int qty2 = i.getQuantity();

qty1+=quantity;

qty2-=quantity;

if(j.getUserId()==1)

holdings.put(j.getStockTicker(), qty1);

if(j.getUserId()==2)

holdings1.put(j.getStockTicker(), qty1);

if(j.getUserId()==3)

holdings2.put(j.getStockTicker(),qty1);

if(j.getUserId()==4)

holdings3.put(j.getStockTicker(),qty1);

if(i.getUserId()==1)

holdings.put(i.getStockTicker(),qty2);

if(i.getUserId()==2)

holdings1.put(i.getStockTicker(),qty2);

if(i.getUserId()==3)

holdings2.put(i.getStockTicker(),qty2);

if(i.getUserId()==4)

holdings3.put(i.getStockTicker(),qty2);

double f1 = traders.get(i.getUserId()).getFunds() + quantity\*askPrice;

traders.get(i.getUserId()).setFunds(f1);

double f2 = traders.get(j.getUserId()).getFunds() - quantity\*askPrice;

traders.get(j.getUserId()).setFunds(f2);

f=1;

}

}

}

}

}

if(f==0){

System.out.println("No transaction executed!");

}

System.out.println();

//printing the stock sector-wise here acc to required output "Pharma"

NSE.getSectorWiseStocks("Pharma");

System.out.println();

//deleting the particular stocks and users acc to required output

NSE.deleteStock("TCS");

traders.remove(1);

System.out.println("Deleted user: Jaydeep");

NSE.deleteStock("M&M");

traders.remove(3);

System.out.println("Deleted user: Kapil");

System.out.println();

//Printing all the stocks registered in the stockExchange

NSE.getAllStocks();

System.out.println();

//Printing all the traders present in the system

System.out.println("Users:");

for(Trader i: traders.values()){

System.out.println(i);

}

//The Market Closes!

System.out.println("\nMarket Closed!");

}

catch ( Exception e){

System.out.println("Error Occurred!!");

e.printStackTrace();

}

}

}

**MAIN CLASS FOR PART 2 OF ASSIGNMENT:**

package com.assignment;

import java.io.File;

import java.util.ArrayList;

import java.util.Scanner;

/\*

notes:

return of a stock is the profit/loss made by buying and selling of share

the max loss a person can make by selling/buying a stock / the lowest price for a given stock during given time is max drawdown

if closePrice>openPrice, stock is bullish that day

if closePrice<openPrice, stock is bearish that day

max-return-potential = net profit earned over the 15 days time period

\*/

public class Part2 {

public static void main(String[] args) {

//starting the question first by extracting the given data for the stock

try{

System.out.println("\nPart2 of Assignment:" + "\n\n" + "Calculating the AveragePrice, MaxDrawdown, MaxReturnPotential: ");

//reading the .csv file

File f = new File("INFY\_15days\_data.csv");

Scanner reader = new Scanner(f);

//declaring the variables to store the extracted data separately

ArrayList<Double> prevClosePrice = new ArrayList<Double>();

ArrayList<Double> openPrice = new ArrayList<Double>();

ArrayList<Double> closePrice = new ArrayList<Double>();

ArrayList<Double> highPrice = new ArrayList<Double>();

ArrayList<Double> lowPrice = new ArrayList<Double>();

ArrayList<Double> lastPrice = new ArrayList<Double>();

String data ="";

int i=0;

//extracting the data and storing it

while(reader.hasNextLine()){

data = reader.nextLine();

if(i==0){

i++;

}

else {

String[] info = data.split(",");

prevClosePrice.add(Double.parseDouble(info[2]));

openPrice.add(Double.parseDouble(info[3]));

highPrice.add(Double.parseDouble(info[4]));

lowPrice.add(Double.parseDouble(info[5]));

lastPrice.add(Double.parseDouble(info[6]));

closePrice.add(Double.parseDouble(info[7]));

i++;

}

}

System.out.println();

//first finding the average price of the stock

double avg = 0.0;

for (Double cp : closePrice) {

avg += cp;

}

double average = Math.floor(avg/closePrice.size()\*100)/100;

System.out.println("Average price of stock over 15 days: " + average);

System.out.println();

//calculating the maxDrawdown of the stock(daily closePrice is the one which is to be considered for the same)

double maxx = closePrice.get(0);

double difference = 0.0;

double val = 0.0;

for (double temp : closePrice) {

if (temp >= maxx) {

maxx = temp;

} else {

difference = maxx - temp;

val = Math.max(difference, val);

}

}

val = Math.floor((val\*100))/100;

System.out.println("Max Drawdown of the given stock: " + val);

System.out.println();

//calculating the max ReturnPotential of the stock using the given data(now: maxReturnPotential for a day will be openPrice-closePrice of that day)

double maxPot = 0.0;

for(int j=0;j<openPrice.size();j++){

maxPot += Math.abs(openPrice.get(j)-closePrice.get(j));

}

maxPot = Math.floor(maxPot\*100)/100;

System.out.println("Max Return Potential for the stock over 15 days: " + maxPot);

System.out.println();

//calculating the maxReturnPotential percentage( {formula used: maxReturnPotential/openPrice on first day}\*100)

double percentage = Math.floor((100\*maxPot)/openPrice.get(0)\*100)/100;

System.out.println("Percentage max Return Potential for the stock is: " + percentage +"%");

}

catch (Exception e){

e.printStackTrace();

System.out.println("Error!!");

}

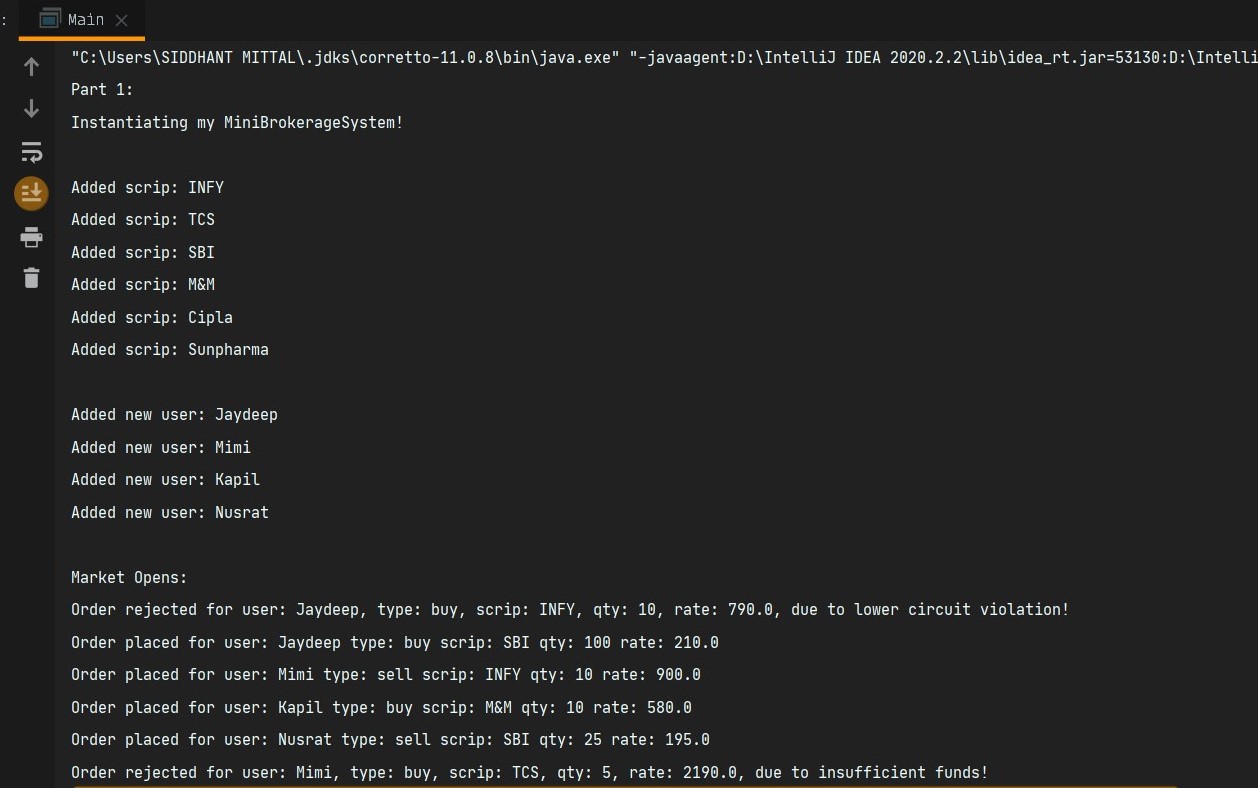
}

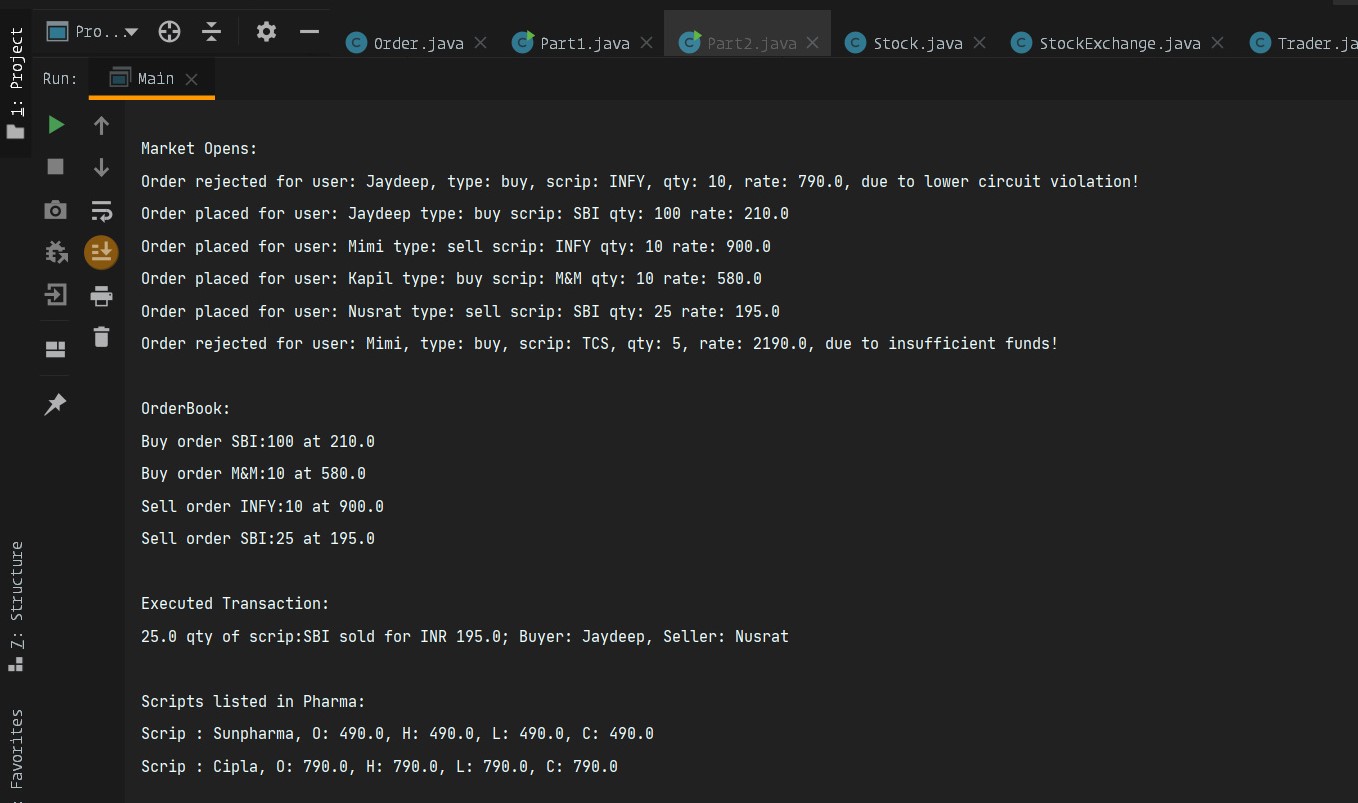
}

\*\* THE CODE ENDS HERE\*\*

**OUTPUT:**

**PART 1:**







**PART 2:**

